Information Security Lab - AS21

Reductions

Handout: Oct 22, 2021 12:00 AM

Due: Nov 8, 2021 3:00 PM

DDH to DLin

Open Task

Reduction of DDH to DLin (25 points)

In this task, your job is to show that if Decisional Diffie-Hellman is hard in a cyclic group $\mathbb{G}$ then the Decision Linear (DLin, described below) assumption is hard, too. To this end, you are expected to write a generic reduction that can decide DDH challenges when given access to an adversary which can decide DLin challenges.

Decision Linear Assumption

Let us first define the DLin game: Let $g \in \mathbb{G}$ be a fixed generator of the cyclic group $\mathbb{G}$ of order p. We define the distribution $\mathcal{L}_{real}$ by:

$\mathcal{L}_{real}$:
Draw $a, b, u, v \leftarrow \mathbb{Z}_p$ uniformly and independently at random.
Output $(g, g^a, g^b, g^{a \cdot u}, g^{b \cdot v}, g^{u+v})$.

And the distribution $\mathcal{L}_{random}$ by:

$\mathcal{L}_{random}$:
Draw $a, b, u, v, w \leftarrow \mathbb{Z}_p$ uniformly and independently at random.
Output $(g, g^a, g^b, g^{a \cdot u}, g^{b \cdot v}, g^w)$.

The challenge of the DLin-assumption is now to distinguish samples of $\mathcal{L}_{real}$ from samples of $\mathcal{L}_{random}$. More precisely, the associated security game works as follows:

1. The DLin-challenger $\mathcal{C}$ draws a random boolean `isReal` $\leftarrow$ { `true`, `false` }.
2. If `isReal` is true, $\mathcal{C}$ draws $(g, g^a, g^b, g^{a \cdot u}, g^{b \cdot v}, g^w) \leftarrow \mathcal{L}_{real}$. Otherwise, it draws $(g, g^a, g^b, g^{a \cdot u}, g^{b \cdot v}, g^w) \leftarrow \mathcal{L}_{random}$. (So, $w = u + v$ does hold iff `isReal` is true.)
3. $\mathcal{C}$ starts a DLin-adversary $\mathcal{A}$ and hands him over the DLin-challenge $(g, g^a, g^b, g^{a \cdot u}, g^{b \cdot v}, g^w)$ when $\mathcal{A}$ asks for it.
4. $\mathcal{A}$ performs some generic computations on $(g, g^a, g^b, g^{a \cdot u}, g^{b \cdot v}, g^w)$ and comes up with a boolean value `solution` which it sends back to $\mathcal{C}$.
5. $\mathcal{C}$ checks if `solution == isReal`. If `solution == isReal` then $\mathcal{A}$ wins. Otherwise, $\mathcal{A}$ loses.

The **advantage** of $\mathcal{A}$ in this security game is defined by
$Adv_{DLin}(\mathcal{A}) := 2 \cdot Pr[\mathcal{A} \text{ wins}] - 1.$
The **DLin-assumption** states that for each ppt adversary $\mathcal{A}$ its advantage in winning the above game is negligible (in the security parameter).

So, the Decision Linear assumption basically states that it is hard to distinguish the sum in the exponent of g^{u+v} from a uniform random group element, even when we know the elements $g, g^a, g^b, g^{a \cdot u}, g^{b \cdot v}$.

To encapsulate DLin challenges in this task, we created a container class which you should use:

```
public class DLin_Challenge<IRandomGroupElement> {
    public final IRandomGroupElement generator;
    public final IRandomGroupElement a;
    public final IRandomGroupElement b;
    public final IRandomGroupElement aTimesU;
    public final IRandomGroupElement bTimesV;
    public final IRandomGroupElement w;
}
```

The Reduction

Your job is to finish the implementation of the class `DDH_DLin_Reduction`:

```
public class DDH_DLin_Reduction extends A_DDH_DLin_Reduction<IRandomGroupElemen
   // your code here
   public Boolean run(I_DDH_Challenger<IRandomGroupElement, IRandomVariable> cha
       // your code here
   }
   public DLin_Challenge<IRandomGroupElement> getChallenge() {
       // your code here
   }
}
```

`DDH_DLin_Reduction` reduces the problem of the DDH assumption to the problem of the DLin assumption. For this end, your reduction needs to decide a `DDH_Challenge`.

For deciding a DDH challenge, the class `DDH_DLin_Reduction` has a field `I_DLin_Adversary<IRandomGroupElement, IRandomVariable> adversary` which contains a perfect DLin Adversary:

```
public interface I_DLin_Adversary<IRandomGroupElement, IRandomVariable> {
     Boolean run(I_DLin_Challenger<IRandomGroupElement,IRandomVariable> challen
}
```

When you call `adversary.run(this)`, the adversary will try to decide a DLin Challenge which you provide. If the DLin challenge which you provide to the adversary contains the group elements $(g, g^a, g^b, g^{a \cdot u}, g^{b \cdot v}, g^w)$, `adversary` will return -- in an overwhelming number of cases -- `true` if $w = u + v$ and `false` otherwise.

Now, given `adversary`, your task is to decide a `DDH_Challenge challenge` which you get from your DDH challenger. `challenge` will consist of four group elements `generator, x, y, z` which can be written as g, g^x, g^y, g^z. You need to return `true` if $z = x \cdot y$ and `false` otherwise.

Randomness

Note that `adversary` is only guaranteed to have an overwhelming advantage when the DLin challenge you provide in `getChallenge()` is distributed according to $\mathcal{L}_{real}$ or $\mathcal{L}_{random}$.

This implies that in each DLin challenge $(g, g^a, g^b, g^{a \cdot u}, g^{b \cdot v}, g^w)$ which you give to `adversary` the exponents a, b, u and v must be drawn uniformly and independently at random. To ensure that this holds we provide you with two fresh independent random variables which you can use to transform your DDH challenge to a correctly distributed DLin challenge.

You can get those random variables by calling:

```
IRandomVariable R1 = challenger.getRandomVariable();
IRandomVariable R2 = challenger.getRandomVariable();
```

You can only get **two** random variables, so do not call `getRandomVariable()` three times or more!

The random variables `R1` and `R2` will be drawn uniformly at random and independent of the exponents of the DDH-challenge which you received by `challenger`. To generate a correctly distributed DLin challenge use exactly the random variables `R1`, `R2` and the elements of the DDH-Challenge which you are given. Do **not** use any other sources of randomness like `java.util.Random` or `java.security.SecureRandom`!

Important Note: To make sure that the DLin challenge $(g, g^a, g^b, g^{a \cdot u}, g^{b \cdot v}, g^w)$ you provide to `adversary` is distributed correctly you need to choose the elements $g, g^a, g^b, g^{a \cdot u}, g^{b \cdot v}$ in a very special way:

- The generator g of the DDH challenge you give to `adversary` must be exactly the generator g of the DDH-challenge which you recevied.
- The exponent a of g^a must be x or y where x, y are the exponents of the elements g^x and g^y of the DDH-Challenge (g, g^x, g^y, g^z) which you recevied.
- The exponent b of g^b must be `R1` or `R2` where `R1` and `R2` are the random variables you received by the challenger.
- While your DLin challenge follows the above rules, it must additionally be as **simple** as possible while being a correctly distributed DLin challenge.

If the DLin-challenge you supply does not comply with the above rules, `adversary` is not guaranteed anymore to have a non-negligible advantage in deciding the DLin-challenge. In this case, the behaviour of `adversary` will be erroneous and the boolean which it will return will be of no help for your reduction.

Tightness

In this task, we are interested in **tight** reductions. This means, your reduction may call `adversary.run` at most once.

Reductions which call `adversary.run` twice or more during a run will only receive **partial** points!

Constructors

Do **under no circumstances** change or remove the constructor of `DDH_DLin_Reduction` which we pre-implemented. The TestRunner needs this empty constructor to test your solution. If this constructor does not exist or work, then the TestRunner can not test your solution and you will receive 0 points.

Testing Your Implementation

To test your implementation, you can use the Run- and Test-Button of the Code-Expert GUI. When you do this, the TestRunner will try to compile your reduction and play the DDH game which we described above several hundred times with it to estimate the advantage of your reduction.

Scores and Points

If the measured advantage is high enough and your reduction is tight, then you should receive full points (25 of 25).

If your reduction is not tight, you will only receive partial points. If your reduction does not follow the rules which we explained here it might have a negligible advantage and will get zero points.

After each run, the TestRunner will tell you how many points your solution got in the *preliminary* tests.

Important Note: The tests which we run in Code Expert are only **preliminary**. After the submission deadline, we will run more exhaustive tests on your solution and review it manually.

Therefore, a solution which is only partially correct may receive full points on Code Expert in the preliminary tests but will get only partial points, eventually. Therefore, make sure that your reductions are correct in the formally theoretic sense of cryptographic reductions!

Time and Memory Restrictions

The resources the TestRunner can use to test your solution are limited. We expect your solution to use less than 10 seconds of CPU time and a restricted space of memory when run several hundred times.

Solutions which run into TimeOut- or OutOfMemoryExceptions will be rejected by us and receive 0 points.

Cheater Warning

The purpose of this task is to algorithmically reduce the decisional Diffie-Hellman problem to the Decision Linear problem.

Any solution which tries to solve the decisional Diffie-Hellman problem by cryptoanalytical algorithms or by "tricking" the testing environment is considered to be a cheating attempt and will receive zero points.

Licensing

[POS19] Stanislav Poslavsky, Rings: An efficient Java/Scala library for polynomial rings, Computer Physics Communications, Volume 235, 2019, Pages 400-413, doi:10.1016/j.cpc.2018.09.005